

LOG4J - LOGGING LEVELS

http://www.tutorialspoint.com/log4j/log4j_logging_levels.htm

Copyright © tutorialspoint.com

The org.apache.log4j.Level class provides following levels but you can also define your custom levels by sub-classing the Level class.

Level	Description
ALL	All levels including custom levels.
DEBUG	Designates fine-grained informational events that are most useful to debug an application.
ERROR	Designates error events that might still allow the application to continue running.
FATAL	Designates very severe error events that will presumably lead the application to abort.
INFO	Designates informational messages that highlight the progress of the application at coarse-grained level.
OFF	The highest possible rank and is intended to turn off logging.
TRACE	Designates finer-grained informational events than the DEBUG.
WARN	Designates potentially harmful situations.

How Level Works?

A log request of level p in a logger with level q, is enabled if $p \geq q$. This rule is at the heart of log4j. It assumes that levels are ordered. For the standard levels, we have ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

Following example makes it clear that how we can filter out all the DEBUG and INFO messages. This program makes use of logger method setLevel(Level.X) to set a desired logging level:

This example would print all the messages except Debug and Info:

```
import org.apache.log4j.*;

public class LogClass {
    private static org.apache.log4j.Logger log = Logger
        .getLogger(LogClass.class);

    public static void main(String[] args) {
        log.setLevel(Level.WARN);

        log.trace("Trace Message!");
        log.debug("Debug Message!");
        log.info("Info Message!");
        log.warn("Warn Message!");
        log.error("Error Message!");
        log.fatal("Fatal Message!");
    }
}
```

When you compile and run LogClass program it would generate following result:

```
Warn Message!
```

```
Error Message!  
Fatal Message!
```

Setting Levels using Configuration File:

Log4j provides you configuration file based level setting which puts you free from changing source code when you want to change debugging level.

Following is an example configuration file which would do the same as we did using *log.setLevel(Level.WARN)* method in the above example.

```
# Define the root logger with appender file  
log = /usr/home/log4j  
log4j.rootLogger = WARN, FILE  
  
# Define the file appender  
log4j.appender.FILE=org.apache.log4j.FileAppender  
log4j.appender.FILE.File=${log}/log.out  
  
# Define the layout for file appender  
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout  
log4j.appender.FILE.layout.conversionPattern=%m%n
```

Now let us use our following program:

```
import org.apache.log4j.*;  
  
public class LogClass {  
    private static org.apache.log4j.Logger log = Logger  
        .getLogger(LogClass.class);  
    public static void main(String[] args) {  
        log.trace("Trace Message!");  
        log.debug("Debug Message!");  
        log.info("Info Message!");  
        log.warn("Warn Message!");  
        log.error("Error Message!");  
        log.fatal("Fatal Message!");  
    }  
}
```

Now compile and run above program and you would get following result in /usr/home/log4j/log.out file:

```
Warn Message!  
Error Message!  
Fatal Message!
```